Towards a (de)composable workflow architecture to define data collection policies

Cyril Cecchinel
Université Nice – Sophia Antipolis
CNRS, I3S, UMR 7271
06900 Sophia Antipolis, France
cecchine@i3s.unice.fr

Sébastien Mosser
Université Nice – Sophia Antipolis
CNRS, I3S, UMR 7271
06900 Sophia Antipolis, France
mosser@i3s.unice.fr

Philippe Collet
Université Nice – Sophia Antipolis
CNRS, I3S, UMR 7271
06900 Sophia Antipolis, France
collet@i3s.unice.fr

ABSTRACT

Sensor networks are classically used in the Internet of Things to collect data, typically supporting Smart Cities or Smart Homes use cases. However, a deep knowledge of these networks is needed to properly develop applications over the deployed systems. This leads to a target mismatch: developers know how to exploit the collected data to develop large-scale "smart" systems, but do not have enough knowledge to technically enact and compose such behaviors on a given sensor network. In this paper, we envision a tooled approach that supports data collection policies management at a higher level of abstraction, fostering reuse. We discuss an architectural abstraction based on workflow concepts assisting developers in expressing data collection policies. The resulting architectures are then composable to be enacted on the same sensor network, but they can also be partially reused through a selection operator.

CCS Concepts

•Software and its engineering → Abstraction, modeling and modularity; *Domain specific languages;* **•Computer systems organization →** *Embedded software;*

Keywords

Sensor network, software architecture, software reuse

1. INTRODUCTION

According to the Gartner group, up to 26 billions of things could be connected to the Internet by 2020. These physical objects are interconnected between each others, forming the *Internet of Things* [7], and creating a mesh of devices producing huge information flows. This mesh is mainly organised around sensor networks deployed in large scale sensing infrastructures, such as *smart cities* or *smart buildings*, which continuously collect data about their environment.

The emergence of sensor networks entails the development of many new applications using these data. The main challenge here is the gap that naturally exists between what a developer knows about the sensor networks and how the relevant data can be collected. For now, the sensor network needs to be configured at the hardware level and programmed at a low level according to the expressed needs. This activity is extremely tedious for developers as they need to understand the diverse architectures of sensor networks and deal with low-level programming languages (*e.g.*, C, nesC) and concepts. Moreover, experts cannot easily reuse, partially or not, data collection policies defined by other experts on different deployments. Abstractions over wireless sensor networks do not cover the expression and reuse of data collection policies [16] while dedicated languages are platform-specific.

This paper aims to highlight an approach dedicated to support developers interacting with sensor networks. We advocate that some architectural abstractions, based on workflow concepts, should be provided to developers. They would enable them to interact with a sensing infrastructure in a more intentional way (*i.e.*, what they expect from the sensor network) instead of working at a lower level (*i.e.*, how such data are retrieved in the infrastructure). Grounded in workflow, the provided architecture have to be complemented by a composition operator, to automate the composition of several policies on the same infrastructure, and by a selection operator, enabling partial reuse of any data collection policy.

2. MOTIVATIONS

According to a recent study [11], each year in France, drivers spend 70 million hours looking for a parking space, which represents 10% of the global traffic and a loss of 700 M€. Detecting illegal parking is also a concern in cities as it creates traffic jams and dangerous situations. To address these kinds of issues, a city may use several applications to monitor parking facilities. Each application relies on one or more data collection policies, *i.e.*, sets of operations performed on data to convert them into knowledge [8]. To illustrate the definition of data collection policies, we chose various Smart Parking application scenarios from different European projects as they are representative of real experiments with deployment in cities (*e.g.*, Santander, Spain) and usage by citizens. The following scenarios, adapted from simplified but non-trivial Smart Parking use cases, rely on data collections from sensor networks:

A. *Information screens.* To help people to park their car where space is available, the city should deploy screens showing the number of available parking spots per district.
B. *Real-time free parking monitoring.* The city should know in real-time the number of free parking spots.

If this amount falls under a threshold, an alert should be displayed on screens located outside the city to encourage citizens to commute.

C. *Availability of disabled parking spaces.* Disabled people should receive notification if adapted parking spots are available in the nearby area. Town-planners want also to know the number of free disabled parking spots to monitor the urban parking policy.

Cities share common concerns and develop their application and data collection policies in their own corner, without taking into account potential reuse between and inside applications, nor between deployed sensing infrastructures. Considering the important investment required by the installation of a sensor network[1], reusing infrastructures is necessary. As data collection policies inside them are at the same time quite complex and fine tuned for a specific deployment, we advocate that reuse should be supported at the level of data collection policies by being able to easily and efficiently use several of them on the same infrastructure, but also by selecting only the relevant part of a given policy for another developer or scenario.

In software engineering, reusability is presented as a key concept allowing *"a better domain architecting and engineering [...] supported both by reuse frameworks and by domain-specific business languages"* [4]. However, these policies are designed by developers unfamiliar to programming specificities inherant to sensing infrastructures (*e.g.*, staged programming, energy-saving programs). To allow developers to express data collection policies on sensing infrastructures, classical approaches rely on the usage of *Domain Specific Languages* (DSL) or graphical languages. Traditional approaches target only one type of platform and do not allow one to deploy the program across multiple platforms distributed over a sensing infrastructure. To reuse a policy over different sensing infrastructure, a developer should use a language adapted to her need abstracting the underlying complexity of such infrastructures. This leads to our first question, **what would be the right level of abstraction for reuse?** *(Q_1)*

Then, the heterogeneity of sensing infrastructures prevents the reuse of the same code or standard processes between different infrastructures, even if the need is the same. For example, both Smart Santander [14] and Butler Smartlife [1] projects propose extremely similar Smart Parking applications, but they both have been developed from scratch. To make the development of wireless sensor network applications easier, a developer should address both data processing-related concerns and network-related concerns [16]. Although there are reuse mechanisms for network-related concerns [16], there is still a lack of approaches in reusing data collection policies. As the redefinition of applications can be time-consuming, error-prone and results in multiple definitions of the same concepts, a developer should be able to reuse data collection policies. But as some part of these previous data collection policies can be irrelevant for her own needs, she should also be able to select only a sub-part of it. Therefore, our second question is **how to enable the reuse of data collection policies?** *(Q_2)*

[1] *e.g.*, 15 M€ have been invested for the Smart Parking infrastructure within the city of Nice, France (http://enstotoday.com/parking-worth-paying-for/).

3. CHALLENGES

3.1 Architectural Abstraction for Data Collection Policies

When a sensor network is deployed, users expect to collect data in order to analyze their environment. However, before collecting data, their needs have to be translated in data collection policies and then deployed at the hardware level in the sensor network.

With the aim of meeting the questions defined above, we believe workflows as a suitable architectural abstraction to define data collection policies for two reasons. First they provide a convenient way to define "*a sequence of activities performed in a business that produces a result of observable value to an individual actor of the business*" [10]. Second, as they can be assimilated to directed acyclic graphs [13], one can use classical graph (de)composition transformations so to allow a developer to reuse entire data collection policies (hierarchical graphs [6]) or parts of them (pruning and weaving [3] operations).

A developer should rely on a DSL leveraging the data-flow perspective of workflow to abstract data collection policies. A policy is then expressed by the developer by refining activities as her own concepts, and dependencies as a data flow between the concepts.

3.2 Reusing a data collection policy

Among the considered scenarios, a developer is likely to deploy a large scale data collection policy, which needs to reuse a previously defined policy deployed in another context or at a smaller scale. As highlighed by question Q_2, she should not have to rewrite the whole policy to do so.

Vision. The hierarchical workflow mechanism – used for example in the Kepler [2] and YAWL [17] workflow languages – allows a workflow designer to encapsulate a workflow into an other one. We plan to leverage this concept to data collection policies with a *Process* concept. A *Process* is then created through a two-steps transformation. First, a new data collection policy is created from the input one, but without any workflow inputs nor outputs. This allows us to separate the data processing from their initial sources and destinations and to connect it to new sensors or collectors. Secondly, this newly created policy is encapsulated into a *Process* concept. This operation as the same number of sources (respectively destination) ports as the original policy sources (respectively destination).

3.3 Reusing parts of a data collection policy

A previously defined policy may embed a data collection pattern that satisfies most of the needs of a developer. Again she should be able to reuse it without having to redesign her policy. Model manipulation operators have been proposed to support such requirement, for example meta-model pruning [15]. This class of algorithm identifies the subset of elements linked to a given one, and proceeds to a clear extraction of the element and the related ones.

Vision. An interesting way to tackle this challenge would be to provide to developers a selection operator σ allowing them to select concepts in a given policy according to the same pruning principles. This operator could defined as follows:

For a subset s of concepts defined in a policy p, the selection operator creates a new policy p' containing only the selected concepts. If two concepts were connected in p, the selection operation keeps this relationship. It must be noted that the σ-operator also removes orphan concepts' ports resulting from the deletion of links.

3.4 Composing data collections policies

To perform the composition between the previous deployed data collection policies and the new ones, we need an operator that works at the workflow level.

Vision. To tackle this challenge, we identify a composition operator ω leveraging the concept of join points introduced in *Aspect-Oriented Programming* [9] to create extension points at the workflow level. In data collection policies, a join point would be defined as a special data collection policy source (respectively destination) where it is possible to connect another destination (respectively source) join point. Each input or output port of an activity can then be extended with a join point as in the work of Mosser et al. on Web services [12].

To perform the composition process, the ω-operator could be defined as follows: Given two policies p_a and p_b, the ω-operator matches destination join points in p_a with source join points in p_b according to an association list l. It returns a new policy p' where each association between a destination join point of an operation a and a source join point of an operation b creates a link l between an output port of a and an input port of b.

4. CONCLUSION & PERSPECTIVES

In this paper, we described a framework that helps developers in defining and reusing data collection policies over sensor networks. The framework relies on architectural abstractions based on workflow concepts, as well as on composition and decomposition operators. The composition one enables several policies to be woven on the same infrastructure, while decomposition allows experts to partially reuse any policy.

Future work will first aim at applying the proposed abstraction and reuse operators on Smart City scenarios and consulting developers to understand how the abstraction and the operators are used (*e.g.*, identify whether the composition operator is more heavily used than the decomposition one) and how they can be complemented. At mid term, we aim at extending the proposed framework by supporting the deployment and execution of the policies over different kinds of sensor networks by using model-to-model transformation techniques [5].

5. REFERENCES

[1] Butler project. http://www.iot-butler.eu/.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *16th ICSSDM*. IEEE, 2004.

[3] U. Assmann and A. Ludwig. Aspect weaving with graph rewriting. In *Generative and Component-Based Software Engineering*, pages 24–36. Springer, 2000.

[4] B. Boehm. A view of 20th and 21st century software engineering. In *28th ICSE*, pages 12–29. ACM, 2006.

[5] C. Cecchinel, S. Mosser, and P. Collet. Software Development Support for Shared Sensing Infrastructures: a Generative and Dynamic Approach. In *14th ISCR*, pages 221–236. Springer, 2015.

[6] G. Engels and A. Schürr. Encapsulated hierarchical graphs, graph types, and meta types. *Electronic Notes in Theoretical Computer Science*, 2:101–109, 1995.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Comp. Syst.*, 29(7):1645–1660, 2013.

[8] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97*. Springer.

[10] P. Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[11] A. Le Fauconnier and E. Gantelet. The time looking for a parking space: strategies, associated nuisances and stakes of parking management in france. In *ETC*, 2006.

[12] S. Mosser, M. Blay-Fornarino, and M. Riveill. Web Services Orchestrations Evolution: A Merge Process for Behavioral Evolution. In R. Morrison, D. Balasubramaniam, and K. E. Falkner, editors, *ECSA 2008*, pages 35–49. Springer, 2008.

[13] W. Sadiq and M. E. Orlowska. Applying graph reduction techniques for identifying structural conflicts in process models. In *Advanced Information Systems Engineering*, pages 195–209. Springer, 1999.

[14] L. Sanchez, J. Galache, V. Gutierrez, J. Hernandez, J. Bernat, A. Gluhak, and T. Garcia. Smartsantander: The meeting point between future internet research and experimentation and the smart cities. In *Future Network Mobile Summit (FutureNetw), 2011*, pages 1–8, June 2011.

[15] S. Sen, N. Moha, B. Baudry, and J. Jézéquel. Meta-model Pruning. In A. Schürr and B. Selic, editors, *12th MODELS 2009*, pages 32–46, 2009.

[16] K. Tei, R. Shimizu, Y. Fukazawa, and S. Honiden. Model-driven-development-based stepwise software development process for wireless sensor networks. *Systems, Man, and Cybernetics: Systems*, 2015.

[17] W. M. van der Aalst and A. H. ter Hofstede. Yawl: yet another workflow language. *Information systems*, 2005.